

# Datenbanken und Datensicherheit

Prof. Dr. rer. nat. Peter Gerwinski

8. Januar 2025

# Datenbanken und Datensicherheit

<https://gitlab.cvh-server.de/pgerwinski/dbs>

## 1 Einführung

## 2 Kurzeinführung Unix

## 3 Kurzeinführung TCP/IP

## 4 Relationale Datenbanken

...

### 4.4 Verknüpfungen von Tabellen

### 4.5 Sichten

### 4.6 Schlüsselfelder

### 4.7 Datensicherung

### 4.8 Aggregate

### 4.9 Transaktionen

### 4.10 Indizierung

### 4.11 Funktionen und Trigger

### 4.12 GUI-Zugriff

### 4.13 Datensicherheit bei Datenbanken

### 4.14 Sonstige Datenbanken

...



Änderungen  
vorbehalten

## 4 Relationale Datenbanken

### 4.4 Verknüpfungen von Tabellen

Problem: Gut angelegte Datenbanken (→ Normalformen) sind stark aufgesplittet.

Wie kann man sie weiterhin effizient benutzen?

Lösung: Verknüpfungen von Tabellen

SQL-Befehl: JOIN

Literatur: z. B. <https://de.wikipedia.org/wiki/SQL>

## 4 Relationale Datenbanken

### 4.4 Verknüpfungen von Tabellen

Was machen wir mit Tabelleneinträgen,  
bei denen die **ON**-Bedingung nicht erfüllt ist?

```
SELECT <Feld[er]> FROM <Tabelle1> [INNER] JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ weglassen: **[INNER] JOIN**

```
SELECT <Feld[er]> FROM <Tabelle1> LEFT JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ linke Tabelle trotzdem anzeigen (mit **NULL**-Einträgen): **LEFT JOIN**  
(analog: **RIGHT JOIN** für rechte Tabelle)

```
SELECT <Feld[er]> FROM <Tabelle1> FULL JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ beide Tabellen trotzdem anzeigen (mit **NULL**-Einträgen): **FULL JOIN**

## 4 Relationale Datenbanken

### 4.5 Sichten

```
SELECT <Feld[er]> FROM <Tabelle1> [INNER] JOIN <Tabelle2>  
    ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;
```

→ Wir betrachten beide Tabellen zusammen als eine große Tabelle.

```
CREATE VIEW <Sicht> AS  
    SELECT <Feld[er]> FROM <Tabelle1> JOIN <Tabelle2>  
        ON <Tabelle1>.<Feld> = <Tabelle2>.<Feld>;  
SELECT <Feld[er]> FROM <Sicht> [WHERE ...];
```

→ Wir sprechen das Ergebnis genau wie eine Tabelle an.

→ Es ist möglich, ohne Verlust an Komfort alle Daten in Normalform zu halten.

# 4 Relationale Datenbanken

## 4.6 Schlüsselfelder

```
CREATE TABLE tabelle1 (  
    id INTEGER,  
    ...  
);  
CREATE TABLE tabelle2 (  
    ...  
    tabelle1_id INTEGER,  
    ...  
);  
ALTER TABLE tabelle1 ADD CONSTRAINT tabelle1_pkey  
    PRIMARY KEY (id);  
ALTER TABLE tabelle2 ADD CONSTRAINT tabelle2_fkey  
    FOREIGN KEY (tabelle1_id) REFERENCES tabelle1 (id);
```

- Dem DBMS mitteilen, welche Felder für **JOIN** vorgesehen sind.
- Das DBMS kann mit auf Konsistenz achten.

## 4 Relationale Datenbanken

### 4.6 Schlüsselfelder

- Dem DBMS mitteilen, welche Felder für **JOIN** vorgesehen sind.
- Das DBMS kann mit auf Konsistenz achten.

Beispiel: Man kann Daten, die noch referenziert werden, nicht löschen.

Alternative: Beim Löschen referenzierter Daten, referenzierende Daten mit löschen.

```
ALTER TABLE tabelle2 ADD CONSTRAINT tabelle2_fkey  
    FOREIGN KEY (tabelle1_id) REFERENCES tabelle1 (id)  
    ON DELETE CASCADE;
```

- Naher Verwandter eines Triggers (kommt gleich)

# 4 Relationale Datenbanken

## 4.7 Datensicherung

```
$ pg_dump --clean -h <Rechner> -U <User> -W <Datenbank>
```

- Ausgabe des gesamten Datenbankinhalts als SQL-Quelltext zur Standardausgabe

→ keine Probleme mit sich evtl. ändernden Binärformaten

- Es ist möglich, den Inhalt direkt in einer Pipe weiterzuverarbeiten (z. B. zu komprimieren).
- Zurückspielen: mit `psql`

```
$ psql -h <Rechner> -U <User> -W <Datenbank> \  
    < <Ausgabe von pg_dump>
```

- analog für **MariaDB**: `mariadb-dump`
- Alternative: Binär-Sicherung (auch zur Laufzeit möglich)  
→ Rücksichern schneller als mit Klartext-Sicherung



## 4 Relationale Datenbanken

### 4.8 Aggregate

Rechenoperationen auf der ganzen Spalte, z. B. Summenbildung:

```
testdb=> SELECT SUM (id) FROM tier;  
sum
```

```
-----  
4143
```

Rechenoperationen: SUM, AVG, MIN, MAX, COUNT

Gruppierung:

```
testdb=> SELECT tierart, MIN (id) AS "Minimum" FROM tier  
        GROUP BY tierart;
```

tierart	Minimum
Spinne	94
Hund	7
Kaninchen	42

Auswahl mit der anzuzeigenden Gruppen mit HAVING (entsprechend WHERE)

# 4 Relationale Datenbanken

## 4.9 Transaktionen

- Ziel: Wahrung der Konsistenz
- Methode: zusammengehörige Aktionen zu einer *Transaktion* zusammenfassen
- Beispiel: Überweisung  
Betrag von einem Konto subtrahieren  
und „gleichzeitig“ zu einem anderen Konto addieren
- Realisierung in PostgreSQL: `BEGIN; ... COMMIT;`
- Realisierung in MariaDB: `START TRANSACTION; ... COMMIT;`
- Abbruch einer Transaktion: `ROLLBACK;` statt `COMMIT;`

# 4 Relationale Datenbanken

## 4.10 Indizierung

- Ziel: Performanzgewinn
- Methode: Spalten, in denen häufig gesucht wird, sortieren
- Beispiel: Datenbank mit Kontaktdaten  
Suche nach Name, Adresse, Telefonnummer
- Realisierung in PostgreSQL und MariaDB:

```
CREATE INDEX <Indexname> ON <Tabellenname>
(
    <Spaltenname>,
    ...
);
```

- Vorteil: schnellere Suche  
(**SELECT**) in indizierten Spalten

Suche nach String:

Zeilen	ohne Index	mit Index
1000	1.809 ms	2.022 ms
1000000	501.546 ms	1.264 ms

# 4 Relationale Datenbanken

## 4.10 Indizierung

- Ziel: Performanzgewinn
- Methode: Spalten, in denen häufig gesucht wird, sortieren
- Beispiel: Datenbank mit Kontaktdaten  
Suche nach Name, Adresse, Telefonnummer
- Realisierung in PostgreSQL und MariaDB:

```
CREATE INDEX <Indexname> ON <Tabellenname>
(
    <Spaltenname>,
    ...
);
```

- Vorteil: schnellere Suche  
(**SELECT**) in indizierten Spalten
- Nachteil: langsames Einfügen/Ändern/Löschen

Suche nach String:

Zeilen	ohne Index	mit Index
1000	1.809 ms	2.022 ms
1000000	501.546 ms	1.264 ms

# 4 Relationale Datenbanken

## 4.11 Funktionen und Trigger

Funktionen:

- **PROCEDURE** entspricht einer **void**-Funktion in C.
- <https://www.postgresql.org/docs/15/sql-createprocedure.html>

Trigger:

- SQL-Standard: <https://www.sqltutorial.org/sql-triggers/>
- PostgreSQL: <https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/>
- Anwendungsbeispiele:
  - Konsistenz
  - Protokollieren
  - Echtzeit-Aktualisierung bei DBMS-Migration

# 4 Relationale Datenbanken

## 4.12 GUI-Zugriff

- Anwendung nutzt DBMS-Client-Bibliothek  
GUI-Programmierung: wie gewohnt
- Spezialfall: Web-Anwendung

Beispiel: Programmiersprache PHP

- Integration in HTML-Quelltext: `<?php ... ?>`
- Objekt zur Kommunikation mit Datenbanken: PDO

Literatur:

- <https://www.postgresqltutorial.com/postgresql-php/connect/>
- <https://www.phptutorial.net/php-pdo/pdo-connecting-to-postgresql/>

## 4 Relationale Datenbanken

### 4.13 Datensicherheit bei Datenbanken

- kein direkter Zugriff von außen auf die Datenbank
- Transportverschlüsselung
- feingranulare Benutzerrechte
- Software aktuell halten
- gegen SQL Injection: Prepared Statements

## 4 Relationale Datenbanken

### 4.13 Datensicherheit bei Datenbanken: SQL Injection

Problem:

- Ein böswilliger Benutzer gibt über eine Benutzerschnittstelle (z. B. ein Web-Interface) Daten ein (z. B. einen „Namen“), die Sonderzeichen enthalten, damit sie als SQL-Befehle ausgeführt werden.
- Beispiele: <https://xkcd.com/327/>, <https://www.heise.de/-10220617>

Lösung: Die Benutzerschnittstelle prüft die Daten auf Sonderzeichen und ersetzt diese durch geeignete Escape-Sequenzen

- ' durch '' ersetzen
- Funktion `CHR ( )`
- Viele DBMS verstehen ein vorangestelltes \.

Bessere Lösung: *Prepared Statements*

- <https://www.postgresql.org/docs/current/sql-prepare.html>
- [https://www.w3schools.com/php/php\\_mysql\\_prepared\\_statements.asp](https://www.w3schools.com/php/php_mysql_prepared_statements.asp)



## 4 Relationale Datenbanken

### 4.14 Sonstige Datenbanken

- Eingebettete Datenbanken:  
Berkeley DB, SQLite  
Software-Bibliothek, keine Client-Server-Struktur
- Nicht-relationale Datenbanken:  
dokumentenorientierte Datenbanken, noSQL  
Performanz wichtiger als Konsistenz  
→ Applikationen stärker in Konsistenzprüfung eingebunden

# 5 Kryptographie

## 5.1 Einführung

### Was ist Datensicherheit?

(CIA)

- Vertraulichkeit (confidentiality) → Verschlüsselung
- Integrität (integrity) → Konsistenzprüfungen, Prüfwerte, Signaturen
- Verfügbarkeit (availability) → Backups, Ausfallsicherheit
  
- Identifizierbarkeit (Authentizität, Nichtabstreitbarkeit, Zurechenbarkeit)  
→ Passwörter, Signaturen  
bzw.
- Anonymität (plausible Abstreitbarkeit, Nichtzurechenbarkeit)  
→ Pseudonymisierung, Anonymisierung,  
Verschlüsselung, Steganographie

# 5 Kryptographie

## 5.1 Einführung

### Was ist Datensicherheit?

- (CIA)
- Vertraulichkeit (confidentiality) → Verschlüsselung
- Integrität (integrity) → Konsistenzprüfungen, Prüfwerte, Signaturen
- Verfügbarkeit (availability) → Backups, Ausfallsicherheit
- Identifizierbarkeit (Authentizität, Nichtabstreitbarkeit, Zurechenbarkeit)  
→ Passwörter, Signaturen  
bzw.
- Anonymität (plausible Abstreitbarkeit, Nichtzurechenbarkeit)  
→ Pseudonymisierung, Anonymisierung,  
Verschlüsselung, Steganographie → Kryptographie

# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: Cäsar-Chiffre, monoalphabetische Substitution

# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: Cäsar-Chiffre, monoalphabetische Substitution, One Time Pad

# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: Cäsar-Chiffre, monoalphabetische Substitution, One Time Pad
  
- Problem: Schlüsselaustausch

# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: Cäsar-Chiffre, monoalphabetische Substitution, One Time Pad, Pseudozufallszahlengenerator, Startwert als Schlüssel
- Problem: Schlüsselaustausch

# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: Cäsar-Chiffre, monoalphabetische Substitution, One Time Pad, *spezielle* Pseudozufallszahlengeneratoren, Startwert als Schlüssel: Enigma, DES, 3DES, RC4, IDEA, Blowfish, TwoFish, CAST, AES, ...
- Problem: Schlüsselaustausch



# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

Unsicher!

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: Cäsar-Chiffre, monoalphabetische Substitution, One Time Pad, *spezielle* Pseudozufallszahlengeneratoren, Startwert als Schlüssel: Enigma, DES, 3DES, RC4, IDEA, Blowfish, TwoFish, CAST, AES, ...
- Problem: Schlüsselaustausch

# 5 Kryptographie

## Kryptographie

- Verschlüsselung: symmetrisch, asymmetrisch, hybrid
- Hashes: Einwegfunktionen, Salt
- Signaturen, Zertifikate
- Schlüsselaustausch

## 5.2 Symmetrische Verschlüsselung

Unsicher!

- Derselbe Schlüssel zum Ver- und Entschlüsseln
- Beispiele: **Cäsar-Chiffre**, **monoalphabetische Substitution**, One Time Pad, *spezielle* Pseudozufallszahlengeneratoren, Startwert als Schlüssel: **Enigma**, **DES**, **3DES**, **RC4**, IDEA, Blowfish, TwoFish, CAST, AES, ...
- Problem: Schlüsselaustausch
- Lösung: *asymmetrische Verschlüsselung*

## 5.3 Asymmetrische Verschlüsselung

- verschiedene Schlüssel zum Ver- und Entschlüsseln:  
öffentlicher und privater Schlüssel
- Prinzip: mathematische Operation,  
einfach durchzuführen, schwer rückgängig zu machen
- Beispiel:  $N = p \cdot q$  – einfacher als Primfaktorzerlegung von  $N \longrightarrow$  RSA  
 $73 \cdot 97 = 7081$ : geht notfalls noch im Kopf  
Primfaktorzerlegung von 7081: mindestens schriftlich, besser mit Rechner
- Beispiel:  $c = b^a$  – einfacher als  $a = \log_b c \longrightarrow$  Diffie-Hellman, ElGamal  
 $7^5 = 16807$ : geht notfalls noch im Kopf  
 $\log_7 16807$ : mindestens schriftlich, besser mit Rechner

→ Details: Algorithmen und Datenstrukturen

## 5.3 Asymmetrische Verschlüsselung

- verschiedene Schlüssel zum Ver- und Entschlüsseln:  
öffentlicher und privater Schlüssel
- Prinzip: mathematische Operation,  
einfach durchzuführen, schwer rückgängig zu machen
- Beispiel:  $N = p \cdot q$  – einfacher als Primfaktorzerlegung von  $N \longrightarrow$  RSA  
 $73 \cdot 97 = 7081$ : geht notfalls noch im Kopf  
Primfaktorzerlegung von 7081: mindestens schriftlich, besser mit Rechner
- Beispiel:  $c = b^a$  – einfacher als  $a = \log_b c \longrightarrow$  Diffie-Hellman, ElGamal  
 $7^5 = 16807$ : geht notfalls noch im Kopf  
 $\log_7 16807$ : mindestens schriftlich, besser mit Rechner

→ Details: Algorithmen und Datenstrukturen

- Nachteil: wesentlich aufwendiger und daher langsamer  
als symmetrische Verschlüsselung

→ *hybride Verschlüsselung*: nur Schlüsselaustausch asymmetrisch,  
eigentliche Verschlüsselung symmetrisch